	Type	L #	Hits	Search Text	DBs
1	BRS	L1	118	garbage and heap$1 and (processor$1 or ((multiple or plurality or number) adj processor$1) or multiprocessor$1 or multi-processor$1) and (shar$3 near6 memory)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB
2	BRS	L2	38	1 and 707/206.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT;
3	BRS	L3	23	2 and synchroniz$6	USPAT; US-PGPUB; EPO; JPO; DERWENT;
4	BRS	L4	650	"23" and mark$3 and relocation and phase	USPAT; US-PGPUB; EPO; JPO; DERWENT;
5	BRS	L5	1	3 and mark$3 and relocation and phase	USPAT; US-PGPUB; EPO; JPO; DERWENT;
6	BRS	L6	1	5 and object$1 near memory	USPAT; US-PGPUB; EPO; JPO; DERWENT;
7	BRS	L7	0	2 and rendezvous	USPAT; US-PGPUB; EPO; JPO; DERWENT;
8	BRS	L8	1	1 and rendezvous	USPAT; US-PGPUB; EPO; JPO; DERWENT;
9	BRS	L9	96	garbage and (regions! or heaps!) and (processors! or ((multiple or plurality or number) adj processor$1) or multiprocessor$1 or multi-processor$1) and (shar$3 near6 memory) and synchroniz$6	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB
10	BRS	L10	9	9 and 707/206.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT;
11	BRS	L11	567	(garbage and (processor$1 or ((multiple or plurality or number) adj processor$1) or multiprocessor$1 or multi-processor$1)).ti,ab.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB
12	BRS	L12	371	(garbage and (processor$1 or ((multiple or plurality or number) adj processor$1) or multiprocessor$1 or multi-processor$1)).ti.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB

	Type	L #	Hits	Search Text	DBs
13	BRS	L13	5	12 and 707/206.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT;
14	BRS	L14	1	12 and (garbage and heap$1 and (processor$1 or ((multiple or plurality or number) adj processor$1) or multiprocessor$1 or multi-processor$1) and (shar$3 near6 memory)).ab.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB
15	BRS	L16	25	15 and mark$3 and relocat$3 and phase$1	USPAT; US-PGPUB; EPO; JPO; DERWENT;
16	BRS	L17	4	15 and mark$3 and relocat$3 and phases! and thread$6	USPAT; US-PGPUB; EPO; JPO; DERWENT;
17	BRS	L19	0	18 and compact$6	USPAT; US-PGPUB; EPO; JPO; DERWENT;
18	BRS	L20	26	12 and compact$6	USPAT; US-PGPUB; EPO; JPO; DERWENT;
19	BRS	L18	3	17 and plan$5	USPAT; US-PGPUB; EPO; JPO; DERWENT;
20	BRS	L15	96	(garbage and (regions! or heaps!) and (processors! or ((multiple or plurality or number) adj processor$1) or multiprocessor$1 or multi-processor$1) and (shar$3 near6 memory)) and synchroniz$6	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB
21	BRS	L21	12	15 and processing adj units!	USPAT; US-PGPUB; EPO; JPO; DERWENT;
22	BRS	L23	0	22 and ((garbage near collect$3) and (regions! or heaps!) and (processors! or ((multiple or plurality or number) adj processor$1) or multiprocessor$1 or multi-processor$1) and (shar$3 near4 memory) and synchroniz$6 and thread$3).ab.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB
23	BRS	L24	0	((garbage near collect$3) and (regions! or heaps!) and (processors! or ((multiple or plurality or number) adj processor$1) or multiprocessor$1 or multi-processor$1) and (shar$3 near4 memory) and synchroniz$6 and	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB

	Type	L #	Hits	Search Text	DBs
24	BRS	L25	0	((garbage near collect$3) and (regions! or heaps!) and (processors! or ((multiple or plurality or number) adj processor$1) or multiprocessor$1 or multi-processor$1) and (shar$3 near4	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB
25	BRS	L26	0	((garbage near collect$3) and (regions! or heaps!) and (processors! or ((multiple or plurality or number) adj processor$1) or multiprocessor$1 or multi-processor$1)).ab.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB
26	BRS	L22	52	(garbage near collect$3) and (regions! or heaps!) and (processors! or ((multiple or plurality or number) adj processor$1) or multiprocessor$1 or multi-processor$1) and (shar$3 near4 memory) and synchroniz$6 and	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB
27	BRS	L27	3	22 and mark$3 and relocat$3 and phases!	USPAT; US-PGPUB; EPO; JPO; DERWENT;
28	BRS	L28	20	22 and mark$3 and relocat$3 and phase$1	USPAT; US-PGPUB; EPO; JPO; DERWENT;
29	BRS	L29	7	28 and plan$4	USPAT; US-PGPUB; EPO; JPO; DERWENT;
30	BRS	L30	7	29 and wait$3	USPAT; US-PGPUB; EPO; JPO; DERWENT;
31	BRS	L31	3	30 and stop$4	USPAT; US-PGPUB; EPO; JPO; DERWENT;

Search Results

Search Results for: [mark and plan and compact and rendezvous<AND>((garbage and heap and processor))]
Found 1 of 103,395 searched. → Rerun within the Portal

Search within Results

GO > Advanced Search > Search Help/Tips

Sort by: Title Publication Publication Date Score Binder

Results 1 - 1 of 1 short listing

1 Garbage collecting the Internet: a survey of distributed garbage collection 77%
Saleh E. Abdullahi , Graem A. Ringwood
ACM Computing Surveys (CSUR) September 1998
Volume 30 Issue 3

Internet programming languages such as Java present new challenges to garbage-collection design. The spectrum of garbage-collection schema for linked structures distributed over a network are reviewed here. Distributed garbage collectors are classified first because they evolved from single-address-space collectors. This taxonomy is used as a framework to explore distribution issues: locality of action, communication overhead and indeterministic communication latency.

Results 1 - 1 of 1 short listing

Search Results

Search Results for: [thread<AND>((multiprocessors and heaps<AND>((mark <near/2> sweep<AND>((garbage))))))]
Found 30 of 103,395 searched. → Rerun within the Portal

Search within Results

rendezvous <near/1> point

GO > Advanced Search : > Search Help/Tips

Sort by: Title Publication Publication Date Score Binder

Results 1 - 20 of 30 short listing

Prev Page 1 2 Next Page

1 An on-the-fly reference counting garbage collector for Java 96%
Yossi Levanoni , Erez Petrank
ACM SIGPLAN Notices , Proceedings of the OOPSLA '01 conference on Object Oriented Programming Systems Languages and Applications October 2001
Volume 36 Issue 11

Reference counting is not naturally suitable for running on multiprocessors. The update of pointers and reference counts requires atomic and synchronized operations. We present a novel reference counting algorithm suitable for a multiprocessor that does not require any synchronized operation in its write barrier (not even a compare-and-swap type of synchronization). The algorithm is efficient and may complete with any tracing algorithm.

2 Real-time concurrent collection on stock multiprocessors 94%
A. W. Appel , J. R. Ellis , K. Li
Proceedings of the SIGPLAN'88 conference on Programming Language design and Implementation June 1988

We've designed and implemented a copying garbage-collection algorithm that is efficient, real-time, concurrent, runs on

commercial uniprocessors and shared-memory multiprocessors, and requires no change to compilers. The algorithm uses standard virtual-memory hardware to detect references to “from space” objects and to synchronize the collector and mutator threads. We've implemented and measured a prototype running on SRC's 5-processor Firefly. It will be straightforward to merg ...

3 A generational on-the-fly garbage collector for Java 94%
Tamar Domani , Elliot K. Kolodner , Erez Petrank
ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN '00 conference on Programming language design and implementation May 2000
Volume 35 Issue 5

An on-the-fly garbage collector does not stop the program threads to perform the collection. Instead, the collector executes in a separate thread (or process) in parallel to the program. On-the-fly collectors are useful for multi-threaded applications running on multiprocessor servers, where it is important to fully utilize all processors and provide even response time, especially for systems for which stopping the threads is a costly operation. In this work, w ...

4 Portable, unobtrusive garbage collection for multiprocessor systems 91%
Damien Doligez , Georges Gonthier
Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages February 1994

We describe and prove the correctness of a new concurrent mark-and-sweep garbage collection algorithm. This algorithm derives from the classical on-the-fly algorithm from Dijkstra et al. [9]. A distinguishing feature of our algorithm is that it supports multiprocessor environments where the registers of running processes are not readily accessible, without imposing any overhead on the elementary operations of loading a register or reading or initializing a field. Furthermor ...

5 Very concurrent mark-&-sweep garbage collection without fine-grain synchronization 90%
Lorenz Huelsbergen , Phil Winterbottom
ACM SIGPLAN Notices , Proceedings of the first international symposium on Memory management October 1998
Volume 34 Issue 3

6 Java without the coffee breaks: a nonintrusive multiprocessor garbage collector 88%
David F. Bacon , Clement R. Attanasio , Han B. Lee , V. T. Rajan , Stephen Smith
ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN'01 conference on Programming language design and implementation
May 2001
Volume 36 Issue 5

The deployment of Java as a concurrent programming language has created a critical need for high-performance, concurrent, and incremental multiprocessor garbage collection. We present the *Recycler*, a fully concurrent pure reference counting garbage collector that we have implemented in the Jalapeño Java virtual machine running on shared memory multiprocessors.

While a variety of multiprocessor collectors have been proposed and some have been implemented, experimental dat ...

7 Creating and preserving locality of java applications at allocation and garbage collection times 87%
Yefim Shuf , Manish Gupta , Hubertus Franke , Andrew Appel , Jaswinder Pal Singh
ACM SIGPLAN Notices , Proceedings of the 17th ACM conference on Object-oriented programming, systems, languages, and applications
November 2002
Volume 37 Issue 11

The growing gap between processor and memory speeds is motivating the need for optimization strategies that improve data locality. A major challenge is to devise techniques suitable for pointer-intensive applications. This paper presents two techniques aimed at improving the memory behavior of pointer-intensive applications with dynamic memory allocation, such as those written in Java. First, we present an allocation time object placement technique based on the recently introduced notion of *p* ...

8 Concurrent replicating garbage collection 87%
James O'Toole , Scott Nettles
ACM SIGPLAN Lisp Pointers , Proceedings of the 1994 ACM

conference on LISP and functional programming July 1994
Volume VII Issue 3

We have implemented a concurrent copying garbage collector that uses replicating garbage collection. In our design, the client can continuously access the heap during garbage collection. No low-level synchronization between the client and the garbage collector is required on individual object operations. The garbage collector replicates live heap objects and periodically synchronizes with the client to obtain the client's current root set and mutation log. An experimental implementation usi ...

9 A concurrent copying garbage collector for languages that distinguish (im)mutable data — 85%

Lorenz Huelsbergen , James R. Larus
ACM SIGPLAN Notices , Proceedings of the Fourth ACM SIGPLAN symposium on Principles & practice of parallel programming July 1993

Volume 28 Issue 7

This paper describes the design and implementation of a concurrent compacting garbage collector for languages that distinguish mutable data from immutable data (e.g., ML) as well for languages that manipulate only immutable data (e.g., pure functional languages such as Haskell). The collector runs on shared-memory parallel computers and requires minimal mutator/collector synchronization. No special hardware or operating system support is required.

10 A scalable mark-sweep garbage collector on large-scale shared-memory machines — 83%

Toshio Endo , Kenjiro Taura , Akinori Yonezawa
Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM) November 1997

This work describes implementation of a mark-sweep garbage collector (GC) for shared-memory machines and reports its performance. It is a simple "parallel" collector in which all processors cooperatively traverse objects in the global shared heap. The collector stops the application program during a collection and assumes a uniform access cost to all locations in the shared heap. Implementation is based on the Boehm-Demers-Weiser conservative GC (Boehm GC). Experiments have been done on Ultra ...

11 Parallel generational garbage collection — 82%

Ravi Sharma , Mary Lou Soffa
ACM SIGPLAN Notices , Conference proceedings on Object-oriented

programming systems, languages, and applications November 1991
Volume 26 Issue 11

12 A parallel, incremental and concurrent GC for servers 80%
Yoav Ossia , Ori Ben-Yitzhak , Irit Goft , Elliot K. Kolodner , Victor Leikehman , Avi Owshanko
ACM SIGPLAN Notices , Proceeding of the ACM SIGPLAN 2002 Conference on Programming language design and implementation May 2002
Volume 37 Issue 5

Multithreaded applications with multi-gigabyte heaps running on modern servers provide new challenges for garbage collection (GC). The challenges for "server-oriented" GC include: ensuring short pause times on a multi-gigabyte heap, while minimizing throughput penalty, good scaling on multiprocessor hardware, and keeping the number of expensive multi-cycle fence instructions required by weak ordering to a minimum. We designed and implemented a fully parallel, incremental, mostly concurrent colle ...

13 Concurrency, Parallelism, Distribution (1): Heap architectures for concurrent languages using message passing 80%
Erik Johansson , Konstantinos Sagonas , Jesper Wilhelmsson
Proceedings of the third international symposium on Memory management June 2002

We discuss alternative heap architectures for languages that rely on automatic memory management and implement concurrency through asynchronous message passing. We describe how interprocess communication and garbage collection happens in each architecture, and extensively discuss the tradeoffs that are involved. In an implementation setting (the Erlang/OTP system) where the rest of the runtime system is unchanged, we present a detailed experimental comparison between these architectures using bo ...

14 Garbage collection for strongly-typed languages using run-time type reconstruction 80%
Shail Aditya , Christine H. Flood , James E. Hicks
ACM SIGPLAN Lisp Pointers , Proceedings of the 1994 ACM conference on LISP and functional programming July 1994
Volume VII Issue 3

Garbage collectors perform two functions: live-object detection and dead-object reclamation. In this paper, we present a new technique for live-object detection based on run-time type reconstruction for a strongly typed, polymorphic language. This

scheme uses compile-time type information together with the run-time tree of activation frames to determine the exact type of every object participating in the computation. These reconstructed types are then used ...

15 The design and implementation of distributed Smalltalk 80%
John K. Bennett
ACM SIGPLAN Notices , Conference proceedings on Object-oriented programming systems, languages and applications December 1987
Volume 22 Issue 12

Distributed Smalltalk (DS) is an implementation of Smalltalk that allows objects on different machines to send and respond to messages. It also provides some capability for sharing objects among users. The distributed aspects of the system are largely user transparent and preserve the reactive quality of Smalltalk objects. Distributed Smalltalk is currently operational on a network of Sun workstations. The implementation includes an incremental distributed garbage collector and support for ...

16 Techniques for obtaining high performance in Java programs 80%
Iffat H. Kazi , Howard H. Chen , Berdenia Stanley , David J. Lilja
ACM Computing Surveys (CSUR) September 2000
Volume 32 Issue 3

This survey describes research directions in techniques to improve the performance of programs written in the Java programming language. The standard technique for Java execution is interpretation, which provides for extensive portability of programs. A Java interpreter dynamically executes Java bytecodes, which comprise the instruction set of the Java Virtual Machine (JVM). Execution time performance of Java programs can be improved through compilation, possibly at the expense of portabili ...

17 A concurrent, generational garbage collector for a multithreaded implementation of ML 80%
Damien Doligez , Xavier Leroy
Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages March 1993

This paper presents the design and implementation of a “quasi real-time” garbage collector for Concurrent Caml Light, an implementation of ML with threads. This two-generation system combines a fast, asynchronous copying collector on the young generation with a non-disruptive concurrent marking collector on the old generation. This design crucially relies on the ML compile-time distinction between

mutable and immutable objects.

18 Real-time replication garbage collection 80%
Scott Nettles , James O'Toole
ACM SIGPLAN Notices , Proceedings of the conference on Programming language design and implementation June 1993
Volume 28 Issue 6

We have implemented the first copying garbage collector that permits continuous unimpeded mutator access to the original objects during copying. The garbage collector incrementally replicates all accessible objects and uses a mutation log to bring the replicas up-to-date with changes made by the mutator. An experimental implementation demonstrates that the costs of using our algorithm are small and that bounded pause times of 50 milliseconds can be readily achieved.

19 Improving the performance of SML garbage collection using application-specific virtual memory management 80%
Eric Cooper , Scott Nettles , Indira Subramanian
ACM SIGPLAN Lisp Pointers , Proceedings of the 1992 ACM conference on LISP and functional programming January 1992
Volume V Issue 1

We improved the performance of garbage collection in the standard ML of New Jersey system by using the virtual memory facilities provided by the Mach kernel. We took advantage of Mach's support for large sparse address spaces and user-defined paging servers. We decreased the elapsed time for realistic application by as much as a factor of 4.

20 The portable common runtime approach to interoperability 77%
M. Weiser , A. Demers , C. Hauser
ACM SIGOPS Operating Systems Review , Proceedings of the twelfth ACM symposium on Operating systems principles November 1989
Volume 23 Issue 5

Operating system abstractions do not always reach high enough for direct use by a language or applications designer. The gap is filled by language-specific runtime environments, which become more complex for richer languages (CommonLisp needs more than C+ +, which needs more than C). But language-specific environments inhibit integrated multi-lingual programming, and also make porting hard (for instance, because of operating system dependencies). To help solve these problems, we have built ...

Results 1 - 20 of 30 **short listing**

Search Results

Search Results for: [garbage and heap and processor and rendezvous]
Found 22 of 103,395 searched. → Rerun within the Portal

Search within Results

GO > Advanced Search > Search Help/Tips

Sort by: Title Publication Publication Date Score Binder

Results 1 - 20 of 22 short listing

Prev Page 1 2 Next Page

1 Task dependence and termination in Ada 84%
Laura K. Dillon
ACM Transactions on Software Engineering and Methodology (TOSEM) January 1997
Volume 6 Issue 1

This article analyzes the semantics of task dependence and termination in Ada. We use a contour model of Ada tasking in examining the implications of and possible motivation for the rules that determine when procedures and tasks terminate during execution of an Ada program. The termination rules prevent the data that belong to run-time instances of scope units from being deallocated prematurely, but they are unnecessarily conservative in this regard. For task instances that are created by i ...

2 CML: A higher concurrent language 82%
John H. Reppy
ACM SIGPLAN Notices , Proceedings of the conference on Programming language design and implementation May 1991
Volume 26 Issue 6

3 Garbage collection and task deletion in distributed applicative 80%

processing systems
Paul Hudak , Robert M. Keller
Proceedings of the 1982 ACM symposium on LISP and functional programming August 1982

The problem of automatic storage reclamation for distributed implementations of applicative languages is explored. Highly parallel distributed systems have several unique characteristics that complicate the reclamation process; in this setting, the deficiencies of existing storage reclamation schemes are thus noted. A real-time, effectively distributed, garbage collector of the mark-sweep variety, called the marking-tree collector, is shown to accomplish reclamation in parallel ...

4 Toward real-time performance benchmarks for Ada 80%
Russell M. Clapp , Louis Duchesneau , Richard A. Volz , Trevor N. Mudge , Timothy Schultze
Communications of the ACM August 1986
Volume 29 Issue 8

Benchmarks are developed to measure the Ada notion of time, the Ada features believed important to real-time performance, and other time-related features that are not part of the language, but are part of the run-time system; these benchmarks are then applied to the language and run-time system, and the results evaluated.

5 Computing curricula 2001 80%
Journal of Educational Resources in Computing (JERIC) September 2001

6 Diverse Topics: Dynamic memory management for programmable devices 80%
Sanjeev Kumar , Kai Li
Proceedings of the third international symposium on Memory management June 2002

The paper presents the design and implementation of a novel dynamic memory-management scheme for ESP---a language for programmable devices. The firmware for programmable devices has to be fast and reliable. To support high performance, ESP provides an explicit memory-management interface that can be implemented efficiently. To ensure reliability, ESP uses a model checker to verify memory safety.The VMMC firmware is used as a case study to evaluate the effectiveness of this memory-management sche ...

7 Can dataflow subsume von Neumann computing? 77%
R. S. Nikhil
ACM SIGARCH Computer Architecture News , Proceedings of the 16th annual international symposium on Computer architecture April 1989
Volume 17 Issue 3

We explore the question: “What can a von Neumann processor borrow from dataflow to make it more suitable for a multiprocessor?” Starting with a simple, “RISC-like” instruction set, we show how to change the underlying processor organization to make it multithreaded. Then, we extend it with three instructions that give it a fine-grained, dataflow capability. We call the result P-RISC, for “Parallel RISC.” Finally, we discuss memory support for such multipr ...

8 Ada for closely coupled multiprocessor targets 77%
A. Cholerton
Proceedings of the conference on Tri-Ada '89: Ada technology in context: application, development, and deployment January 1989

The techniques for cross-compiling real-time Ada programs for embedded targets are well developed. Generally, these toolsets enable the user to compile and build a program on the host, load it into the target's memories via some form of serial or parallel link, and then run and debug the program under intensive control from the host. This technology has now been extended by SD to provide similar facilities for a class of closely coupled multiprocessor targets comprising homogeneo ...

9 The Howitzer improvement program: lessons learned 77%
D. Krantz
Proceedings of the conference on Tri-Ada '89: Ada technology in context: application, development, and deployment January 1989

10 Distributed operating systems 77%
Andrew S. Tanenbaum , Robbert Van Renesse
ACM Computing Surveys (CSUR) December 1985
Volume 17 Issue 4

Distributed operating systems have many aspects in common with centralized ones, but they also differ in certain ways. This paper is intended as an introduction to distributed operating systems, and especially to current university research about them. After a discussion of what constitutes a distributed operating system and how it is distinguished from a computer network, various key design issues are discussed. Then several examples of current research projects are examined in some detail ...

11 Parallel execution of prolog programs: a survey 77%
Gopal Gupta , Enrico Pontelli , Khayri A.M. Ali , Mats Carlsson , Manuel V. Hermenegildo
ACM Transactions on Programming Languages and Systems (TOPLAS) July 2001
Volume 23 Issue 4

Since the early days of logic programming, researchers in the field realized the potential for exploitation of parallelism present in the execution of logic programs. Their high-level nature, the presence of nondeterminism, and their referential transparency, among other characteristics, make logic programs interesting candidates for obtaining speedups through parallel execution. At the same time, the fact that the typical applications of logic programming frequently involve irregular computatio ...

12 Issues in optimizing Ada code 77%
David Rosenfeld , Mike Ryer
ACM SIGAda Ada Letters , Proceedings of the working group on Ada performance issues 1990 January 1990
Volume X Issue 3

13 Garbage collecting the Internet: a survey of distributed garbage collection 77%
Saleh E. Abdullahi , Graem A. Ringwood
ACM Computing Surveys (CSUR) September 1998
Volume 30 Issue 3

Internet programming languages such as Java present new challenges to garbage-collection design. The spectrum of garbage-collection schema for linked structures distributed over a network are reviewed here. Distributed garbage collectors are classified first because they evolved from single-address-space collectors. This taxonomy is used as a framework to explore distribution issues: locality of action, communication overhead and indeterministic communication latency.

14 A comparison of the concurrency features of Ada 95 and Java 77%
Benjamin M. Brosgol
ACM SIGAda Ada Letters , Proceedings of the annual ACM SIGAda international conference on Ada November 1998
Volume XVIII Issue 6

15 Very concurrent mark-&-sweep garbage collection without 77%

fine-grain synchronization
Lorenz Huelsbergen , Phil Winterbottom
ACM SIGPLAN Notices , Proceedings of the first international symposium on Memory management October 1998
Volume 34 Issue 3

16 The Flux OSKit: a substrate for kernel and language research 77%
Bryan Ford , Godmar Back , Greg Benson , Jay Lepreau , Albert Lin , Olin Shivers
ACM SIGOPS Operating Systems Review , Proceedings of the sixteenth ACM symposium on Operating systems principles October 1997
Volume 31 Issue 5

17 Ada, C, C++, and Java vs. the Steelman 77%
David A. Wheeler
ACM SIGAda Ada Letters July 1997
Volume XVII Issue 4

This paper compares four computer programming languages (Ada95, C, C++, and Java) with the requirements of "Steelman", the original 1978 requirements document for the Ada computer programming language. This paper provides a view of the capabilities of each of these languages, and should help those trying to understand their technical similarities, differences, and capabilities.

18 Ada development system technical and performance requirements (with rationale) 77%
Donald G. Krantz
Proceedings of the conference on TRI-ADA '90 December 1990

This paper discusses requirements for Ada1 compilers and associated tools used for real-time embedded weapons systems (EWS) development. The requirements have been developed over a period of several years by embedded systems developers at Honeywell Inc. and Alliant Techsystems Inc. Requirements for the run time system, compiler-generated code, and host tools such as linkers are presented. A short rationale statement is provided with each specific requirement.

19 Towards an active network architecture 77%
David L. Tennenhouse , David J. Wetherall
ACM SIGCOMM Computer Communication Review April 1996
Volume 26 Issue 2

Active networks allow their users to inject customized programs

into the nodes of the network. An extreme case, in which we are most interested, replaces packets with "capsules" - program fragments that are executed at each network router/switch they traverse.Active architectures permit a massive increase in the sophistication of the computation that is performed within the network. They will enable new applications, especially those based on application-specific multicast, information fusion, a ...

20 A fine-grained parallel completion procedure 77%
Reinhard Bündgen , Manfred Göbel , Wolfgang Küchlin
Proceedings of the international symposium on Symbolic and algebraic computation August 1994

We present a parallel Knuth-Bendix completion algorithm where the inner loop, deriving the consequences of adding a new rule to the system, is multi-threaded. The selection of the best new rule in the outer loop, and hence the completion strategy, is exactly the same as for the sequential algorithm. Our implementation, which is within the PARSAC-2 parallel symbolic computation system, exhibits good parallel speedups on a standard multi-processor workstation.

IEEE HOME | SEARCH IEEE | SHOP | WEB ACCOUNT | CONTACT IEEE

Membership Publications/Services Standards Conferences Careers/Jobs

IEEE Xplore™ RELEASE 1.4

Welcome
United States Patent and Trademark O

Help FAQ Terms IEEE Peer Review Quick Links

Welcome to IEEE Xplore

- Home
- What Can I Access?
- Log-out

Tables of Contents

- Journals & Magazines
- Conference Proceedings
- Standards

Search

- By Author
- Basic
- Advanced

Member Services

- Join IEEE
- Establish IEEE Web Account

Print Format

SEARCH RESULTS [PDF Full-Text (200 KB)] NEXT DOWNLOAD CITATION

Predicting scalability of parallel garbage collectors on shared memory multipr

Endo, T. Taura, K. Yonezawa, A.
Dept. of Inf. Sci., Tokyo Univ., Japan

This paper appears in: Parallel and Distributed Processing Symposium., Proce International
On page(s): 6 pp.
23-27 April 2001
San Francisco, CA, USA
2001
ISBN: 0-7695-0990-8
Number of Pages: 0
References Cited: 12
INSPEC Accession Number: 6970805

Abstract:
This paper describes a performance prediction model of parallel mark-sweep ç collectors (GC) on shared memory multiprocessors. The prediction model take snapshot and memory access cost parameters (latency and occupancy) as inp outputs performance of the parallel marking on any given number of processc several factors Mat affects performance into account: cache misses costs, mei access contention, and increase of misses by parallelization We evaluate this ı comparing the predicted GC performance and measured performance on two architecturally different shared memory machines: Ultra Enterprise 10000 (cr connected SMP) and Origin 2000 (hypercube connected DSM). Our model acc predicts qualitatively different speedups on the two machines that occurred in application, which turn out to be due to contentions on a memory node. Lit ac performance analysis, applications of the proposed model include adaptive GC to achieve optimal performance based on the prediction. This paper shows the automatic regulation of GC parallelism.

Index Terms:
storage management; shared memory systems; performance evaluation; par garbage collectors; shared memory multiprocessors; performance prediction r performance prediction; parallel mark-sweep

Documents that cite this document
Select link to view other documents in the database that cite this one.

SEARCH RESULTS [PDF Full-Text (200 KB)] NEXT DOWNLOAD CITATION

Evaluation of parallel copying garbage collection on a shared-memory multipr

Imai, A. Tick, E.
Inst. for New Generation Comput. Technol., Tokyo, Japan

This paper appears in: Parallel and Distributed Systems, IEEE Transactions on
On page(s): 1030 - 1040
Sept. 1993
Volume: 4 Issue: 9
ISSN: 1045-9219
References Cited: 22
CODEN: ITDSEO
INSPEC Accession Number: 4582750

Abstract:
A parallel copying garbage collection algorithm for symbolic languages execut shared-memory multiprocessors is proposed. The algorithm is an extension of sequential algorithm with a novel method of heap allocation to prevent fragm and facilitate load distribution during garbage collection. An implementation c algorithm within a concurrent logic programming system, VPIM, has been eva the results, for a wide selection of benchmarks, are analyzed here. The authoi how much the algorithm reduces the contention for critical sections during ga collection, 2) how well the load-balancing strategy works and its expected ove and 3) the expected speedup achieved by the algorithm.

Index Terms:
parallel copying; garbage collection; shared-memory multiprocessor; symboli languages; heap allocation; fragmentation; load distribution; concurrent logic programming system; VPIM; contention; load-balancing; logic programming; algorithms; resource allocation; shared memory systems; storage manageme

Documents that cite this document
Select link to view other documents in the database that cite this one.

A shared-memory multiprocessor garbage collector and its evaluation for committed-choice logic programs

Imai, A. Tick, E.
Icot, Tokyo, Japan

This paper appears in: Parallel and Distributed Processing, 1991. Proceedings Third IEEE Symposium on
On page(s): 870 - 877
2-5 Dec. 1991
Dallas, TX, USA
1991
ISBN: 0-8186-2310-1
Number of Pages: xvi+903
References Cited: 14
INSPEC Accession Number: 4368135

Abstract:
A parallel copying garbage collection algorithm for symbolic languages execut shared-memory multiprocessors is proposed. The algorithm is an extension o sequential algorithm with a novel method of heap allocation to prevent fragm and facilitate load distribution during garbage collection. An implementation c algorithm within a concurrent logic programming system, VPIM, has been eva the results, for a wide selection of benchmarks, are analyzed. The authors sh much the algorithm reduces the contention for critical sections during garbag (2) how well the load-balancing strategy works and its expected overheads, a expected speedup achieved by the algorithm.

Index Terms:
shared-memory multiprocessor garbage collector; committed-choice logic pro parallel copying garbage collection algorithm; symbolic languages; sequential heap allocation; VPIM; benchmarks; load-balancing; logic programming; para algorithms; shared memory systems; storage management

Documents that cite this document
Select link to view other documents in the database that cite this one.